NPS-52PW73081A

NAVAL POSTGRADUATE SCHOOL

Monterey, California

CONTROL STRUCTURES

IN

DIGITAL PROCESSES

by

V. MICHAEL POWERS

29 AUGUST 1973

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral M. B. Freeman
Superintendent

M. U. Clauser
Provost

ABSTRACT:

The control space of a digital process can be viewed as a projection of the state space of the processor. This state space may be an interpretation of some underlying (perhaps physical) processor's state space. A control operator is a projection of a process step: the portion which specifies the "next control state". A set of elementary control structures is defined and used as a common basis for comparing the control structures in a microcomputer and several programming languages. The relationship of this view of control to several areas of computer science research is noted.

The work reported herein was supported in part by the Naval Electronics Laboratory Center under Work Request WR-2-9104 dated 26 April 1972.

NPS-52PW73081A

29 August 1973

CONTROL STRUCTURES IN DIGITAL PROCESSES

TABLE OF CONTENTS

List of Figures

List of Tables

CONTROL STRUCTURES IN DIGITAL PROCESSING

1. Introduction

This paper is directed toward examination of the control portions of digital systems. Algorithms for manipulation of data abound; there is a growing list of functional data operator modules which are being designed and built as modular components for use in special purpose logic systems designs [4]. The design of a general purpose digital computer is related to this work in the sense that the special purpose of such a logic system is to execute sequences of members of a set of instructions which is effectively complete with respect to digital computation.

We shall not assume a priori that the section of a computer historically called the "control unit" completely characterizes the control problem in digital system design. We shall instead attempt to develop a more general and universal notion of control which agrees in large part with such intuitive descriptions. Section 2 is devoted to a brief examination of necessary background in the development of state models of digital processes. Section 3 uses this background to outline a characterization of control.

Although the construction of general purpose digital computers is not the target application of this study, the field does reveal interesting applications. A vital characteristic of our view of structure in control is that

it apply at so-called "high levels" of system description. To this purpose, Section 4 lists a number of examples, mostly computers and languages, of system description at different levels.

Finally, Section 5 briefly addresses several areas of study which are not usually considered part of logic system design, but which appear to have something to lend to, or borrow from, the current discussion of digital system control.

2. The art of the states

Much of the work in this section deals with topics related to ideas summarized and integrated in a recent paper by Horning and Randell [5]. Their terms are freely borrowed and distorted in this section.

The problem of accurately characterizing digital processing systems is difficult. Men can build a very large and complex systems which are usable, although satisfactory measurement and evaluation of the internal operations of these systems is often beyond our present capability. Verification of a design, or proof of correctness, is often impractical in terms of difficulty or length. Consequently, design of effective large systems is usually heuristic and intuitive.

Where design and evaluation methods for large systems have shown promise of becoming effective, they have been based on sound techniques of modularization. Whereas building a system out of a small number of well-defined building blocks has important benefits in the management of the design process, in documentation of the design, and in the logistics and maintenance required to support the system [8], such modularization has its most striking effect on the design process where it reduces the apparent complexity of the system. Modularity, by partitioning the details and characterizing subsets of detail as modules, makes a design easier to understand. Our discussion of control will be mostly concerned with such modular systems.

One of the difficulties encountered in the description of digital processing systems is the magnitude of the number of combinations of conditions which can occur. One of the smaller minicomputers is the PDP-8 with 12-bit words. A minimal configuration has 4096 words of main memory, plus CPU word registers AC (accummulator) and PC (program counter) and bit flipflops L (link), Run and Interrupt state [12]. Other temporary memories in the CPU describe progress during the execution of a given instruction. Between instruction executions, the $12(2^{12} + 2) + 3$ bits of memory might have any one of $_{2}12(2^{12} + 2) + 3 > 32 \times 10^{12}$ combinations of binary values. Any one of these combinations uniquely describes the next instruction to be executed, and consequently the next combination of bit values to be expected (assuming no external interruptions).

The strength and the weakness of describing a digital system in terms of all the possible combinations of values held in its memory unit are these: This description is complete enough to describe the future behavior of the machine, if undisturbed by external events, and this description can (conceptually) be verified by straight forward measurement. But maintaining such a description, much less manipulating it or displaying it, requires a computational effort well beyond the capability of the machine being described, and usually beyond the comprehension of the human observer.

The key to human understanding of such complex systems lies in the identification of "important" variables and values. To the assembly-language programmer writing or tracing a single program step, the important values are those of the PC and the indicated memory word interpreted as an instruction with opcode, flags and address portions. Sometimes the content of possible range of contents of the AC and data word are important; sometimes not. Most of the memory contents have no effect or immediate relevance to the consequences of a single instruction execution.

An analogy may prove suggestive. For most human purposes, a ship's position is plotted in two dimensions. An inertial navigation sensor on the ship may measure position in three dimensions; optical and electromagnetic sensors give coordinates relative to deck orientation as well as ship position. For plotting purposes, however, readings are converted to the coordinates of the plot. Transformations are necessary. In particular, the momentary elevation of the ship on the crest or trough of a swell is ignored: the three-dimensional position of the ship is projected onto the two-dimensional coordinates of the plotting board.

The usual description of a digital logic system is in terms of the binary variables which describe the stable states of the memory elements. A specification of the value of every variable describes a state of the system. The set of states describable by values of variables is the state

space of the system. Actions (calculations of computers) are sequences of states describing a path through the state space.

The useful part of such a fundamental but complex description of the system results from a number of transformations of state variables and a projection of the state space onto a smaller space. The meaningful description to an assembly language programmer of a PDP-8's path among the more than 3×10^{13} states of its space consist of the sequence of addresses of instructions (a projection to the set of 4096 addresses); the interpretations (via transformations) of instructions into operations, modes and addresses; and sometimes, the values of certain selected words of memory. For his purposes, the action of the machine is mapped from the original state space to a smaller space. This mapping, which typically includes projections as well as other transformations, is an interpretation from the space describing the bit patterns to the space describing instructions and addresses.

Other interpretations are used at other levels of design. A FORTRAN programmer is interested in the (conceptual) interpretation in terms of program statements and subroutine calls executed, and variable values changed. His interpretation can be said to map the general purpose binary computer to a FORTRAN machine. The designer of the operating system of a large computer system is often more concerned with an interpretation of "job steps" which might include a program execution as a single step. His interpretation of the hardware

description highlights not bit values or program variable values but the status of resource allocations and scheduling variables.

More detailed discussion of examples is found in a later section. The important facts at this time are that interpretations help capture the portion of a state description which are important at a given level, and that there are several different levels and different interpretations possible. Precise specification of an interpretation involves, among other things, specification of the states and state variables of the image space. The choice of these states and their representation is, as implied above, an imprecise art. We can, however, proceed to discuss control of digital processing systems assuming that it is desirable and sometimes possible to define a state space for an interpretation of basic physical descriptions of a machine.

3. Control in digital processing

3.1 Definition

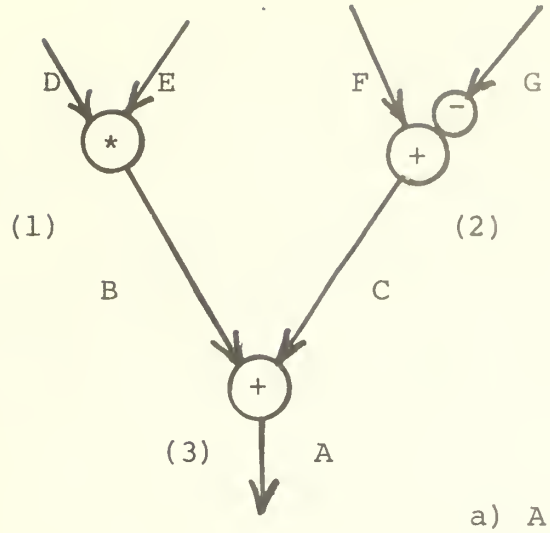
A digital process is a pattern of state changes. For example, a single addition A = B + C is the step between the states which characterize the "old" and the "new" values of variable A.

In a general purpose digital computer, the pattern is channeled into a single sequence; one step is accomplished at a time. Often, the problem being solved has a high degree of parallelism. B may be specified as a product and C a difference. Even with a nondeterministic language specification of the program, in a given execution of the program on a general purpose computer, one of those computational steps proceeds the other. One of the possible sources of improved efficiency in a hardware or microprogrammed implementation is that parallel steps can be executed simultaneously if the necessary data operators and data flow paths are available.

The example of the portion of a calculation,

```
B = D*E     (1)
C = F-G     (2)
A = B+C     (3)
```

is sketched in Figure 3.1.a as a process graph. For users of the computation, the data and their values are the essential parts. For designers of the implementation, management of the computing resources is the job at hand. A means must be adopted to describe progress through the



a) A partial process graph

```
R1 ← D * E
R2 ← F - G
R3 ← R1 + R2
A ← R3
```

c) an alternate program implementation

```
AC ← D
AC ← AC * E
B ← AC
AC ← F
AC ← AC - G
C ← AC
AC ← B
AC ← AC + C
A ← AC
```

b) a program implementation

Figure 3.1 a portion of a calculation

process graph and to control that progress.

With respect to a process graph, the control state variables are a set of state variables sufficient to describe the progress of an execution. The control states are the states defined by the possible values of the control state variables. In a given process graph, the control state identifies which of the program steps have been executed. For an implementation on a deterministic machine, the specification of the next step is inherent in the program. A serial organization is most common. For this reason, a single index is a very important part of the control state. The number in the program counter is the strongest clue of progress in the machine language-level description of a computation. In a FORTRAN machine interpretation, the ISN is the appropriate index. This index alone is usually not sufficient. If the current position within a subroutine is important, so are the values of indexing parameters of a loop and the "caller" of the subroutine. In a timesharing system with virtual memory, the control state must identify not only the current virtual address but also the current process status including the address translation map and status of allocatable resources such as I/O facilities.

Practical methods of performing a computation use only a finite number of devices (most often, one). Each device performs only one step at a time. This device may be the CPU of a digital computer, which loads, stores, adds, subtracts, etc. Or it may be a conceptual device which performs more complicated steps. Examples are a "FORTRAN

machine" or a BASIC or APL interpreter. In any event, one necessary part of forming a program to realize the algorithm of the process graph is to specify the order of steps, within the bounds of the available data paths, data operators, and control operators. Thus, Figure 3.1.b shows a particularly inept programming, but one which is simply derived, of the computation of Figure 3.1.a for a one-address computer with a single accumulator. Figure 3.1.c shows an alternate program for a computer which can specify two input operands and one of several results registers, or a store, in a single step.

In the remainder of this report, the command for the execution of a program step will be considered to be divided into two parts, the data operator and the control. The data operator specifies what manipulation of data is to occur. It is the function which maps from the "old" data values in a state description to the "new". In the example of Figure 3.1, the data operator in step 3 specifies that A is to become the sum of B and C. The control specifies how the control state information changes. In this same example, the "current" indication must change from (2) to (3) while the sum is formed.

The distinction between the data operator and control of course depends heavily on the level of the interpretation. In a vertical microprogramming implementation, the statement (3) above may become the sequence:

. gate register B to adder input 1

. gate register C to adder input 2

. gate adder output to register A

In this implementation the data operators become register transfers or bussing operations and the control becomes a sequencing of these. At a higher-level interpretation of the process the execution of an entire program may map into a single step, and the data operator and control of the paragraph above get absorbed as part of a single operator.

Given, then, an interpretation of a process in terms of states; and given a description of the state variables which distinguish data from control state variables, the following definitions are used to the extent that they apply:

data operators are the portions of the state transition functions which specify data values in the next state;

control operators are the portions of the state transition functions which specify control variable values in the next state.

Fortunately, most implementations and their usual interpretations allow such a partitioning of the state transition function. Usually, two separate functions can be written for data and control operators.

3.2 Control structures

All of the control features listed in this report can be implemented by the five control structures illustrated in the combined diagram of Figure 3.2. This minimal set we will call elementary control structures.

a) start A node with one exit path. The node need not have an associated data operator.

b) step A node with one entry and one exit path. This is the "workhorse" of calculations; most important data operators appear in single steps.

c) branch A node with one entry path and two exit paths. The path taken by a computation depends on the (binary) result of a rest, usually on data values.

d) merge A node with two entry paths and one exit path. Either path may form part of an execution. It is convenient to allow a data operator at this node, but also to allow the null operator. This structure is useful in synchronizing parallel paths or merging of alternate paths.

e) stop A node with one entry path only. Indicates termination of the execution.

Although these five structures are a minimal set, it is often convenient to use complexes such as those in Figure 3.3. There, each of the following are illustrated as they could be implemented with elementary control structures:

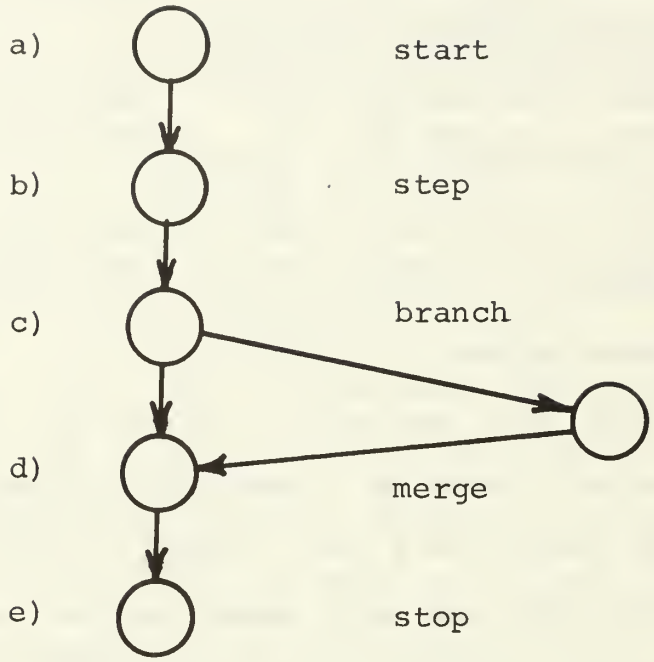
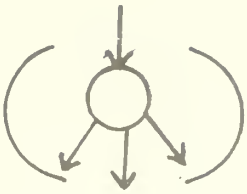
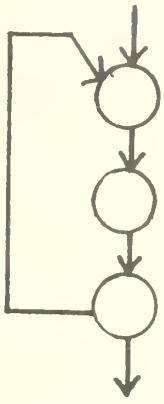


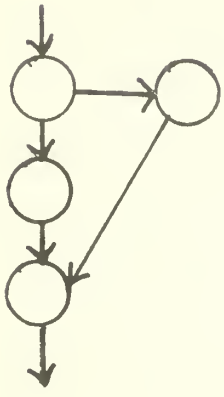
Figure 3.2 Elementary control structure

a) 3-way branch





(b) loop



(c) if then

d) subroutine

Figure 3.3 Control Structure Complexes

a. multiple branch For example, Figure 3.3 part (a) would implement a 3-way branch.

b. loop The principle component is a branch which branches backward or forward depending on indices or some data conditions.

c. if then One of several different complexes of branches, steps and merges.

d. subroutine Encapsulates a portion of a process graph as a single node.

3.3 Interpretation

For computing machines constructed along the conventional "von Neumann machine" lines, the definitions above can usually be identified with the functions of that part of the hardware conventionally called "the control". In Figure 3.4, one of the four parts of the simple conventional computer structure is "... an organ, called the control, which can automatically execute the orders ..." stored in the memory [2].

The definitions in this report identify the operations of the "control unit" of Figure 3.4 as control steps only under a certain sort of interpretation (level of modeling). The control unit of Figure 3.4 performs the elementary control functions only when the interpretation is at the assembly language level, where the control states are identified with the outputs of the control unit and data

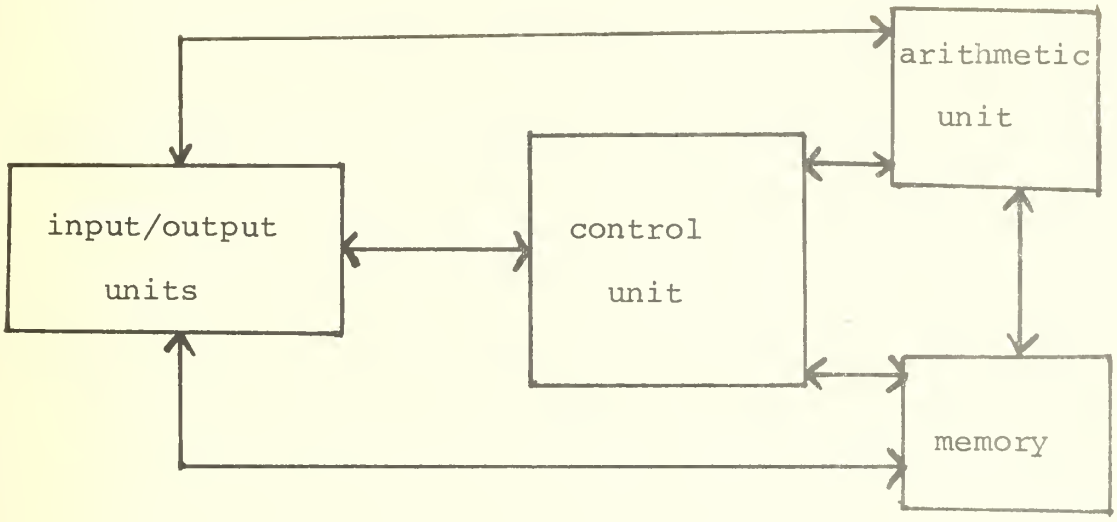


Figure 3.4 A simple conventional computer structure

operations are the hardware operations of the arithmetic unit. The same machine can execute compiled FORTRAN object code. In the FORTRAN interpretation of control states, a state variable corresponding to the Internal Statement Number (ISN) of the executable statement is preeminent. This variable does not (usually) even appear in compiled object code! A single step in this interpretation generally corresponds to a long sequence of steps for the "control unit" of Figure 3.4. As a vehicle for executing the program of a FORTRAN machine, Figure 3.4 shows an organization whose "control unit" uses too little and too much of the control information. It operates with a set of states which is not appropriate for the interpretation.

4. Example Control Structures

This section lists separately several examples of machines or languages which appear to have interesting control structures. The examples are not intended to fully explain the languages, but merely to demonstrate the strong similarities and frequent major differences in control structures among the examples.

Example 1. MCS-4

The MCS-4 microcomputer is a 4 bit CPU configurable with memory and I/O ports. Although the data operations available are minimal, the system compensates by offering a set of control functions which is quite comprehensive, considering the size of the machine. In applications, the MCS-4 competes more with random logic control devices than with minicomputers.

For an interpretation of the MCS-4 operation in terms of single instruction execution the control state can be precisely defined as the content of the following internal registers (an interpretation in terms of memory cycles demands more internal data to specify the control state):

- the contents and current pointer value of the address stack (4 x 12 + 2 = 50 bits)
- the contents of the index registers (16 x 4 = 64 bits)
- the contents of the command control register (variable number of significant bits, depending on the memory size)
- the selection status of each memory chip (1 bit each)

The basic instruction set of the MCS4 4004 (CPU) can be partitioned in the following Control Sets (CSs) according to how they affect the control state. Diagrams describing the control structures involved appear in Figure 4.1.

CS 1 simple step instructions

These instructions usually include a data operation, and involve incrementing the current program counter by 1

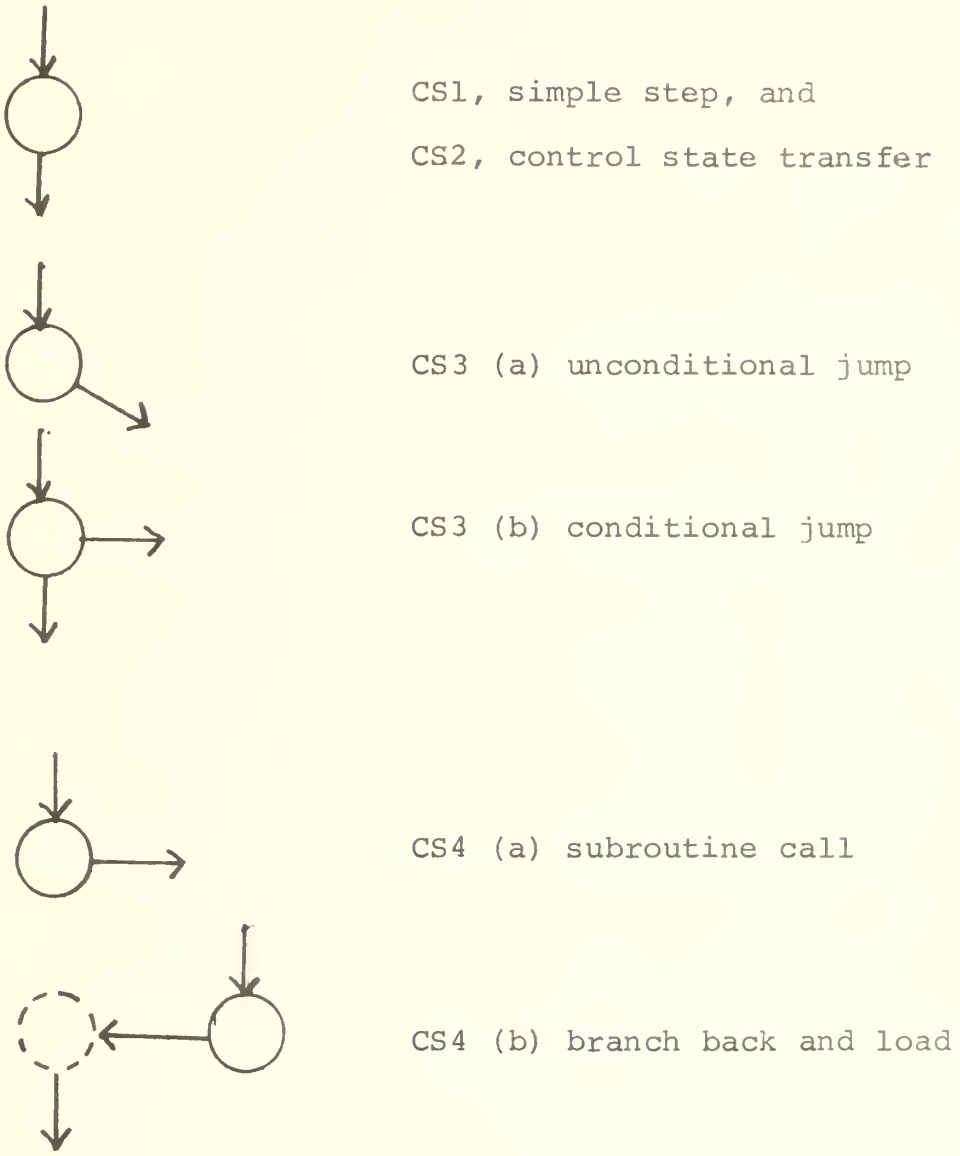


Figure 4.1 Basic control structures in the MCS-4

(if a one-word instruction) or 2 (if a two-word instruction). Included in this class are

machine instructions:

NOP, FIM, FIN, INC, ADD, SUB, LD, LDM

I/O-RAM instructions (all):

WRM, WMP, WRR, WR0, WR1, WR2, WR3, SBM, RDM, RDR, ADM, RD0, RD1, RD2, RD3

accumulator group instructions:

CLB, CLC, IAC, CMC, CMA, RAL, RAR, TCC, DAC, TCS, STC, DAA, KBP

CS 2 control state transfer

a) XCH exchange contents of accumulator with contents of an index register

b) i/o control: use index register contents to set I/O status:

SRC send register control

DLC designate command line

CS 3 jumps

a) unconditional jumps

JUN (jump, unconditional)

JIN (jump, indirect)

b) conditional jumps

JCN (jump on condition) Tests among three different one-bit data conditions

ISZ (increment and continue if zero) Increment a register and jump if the result is not zero. Note

that this involves both data operation and interrogation of control state.

CS 4 stack operations

a) JMS jump to subroutine. Stack (incremented) program counter by incrementing "current" pointer value and loading transfer address into new program counter. Note: the stack over-writes old values starting with the 4th successive stack push operation (4th level of subroutine nesting)

b) BBL branch back and load. Pop the stack (move the pointer to the most recently saved instruction counter) and transfer 4 bits of data to the accumulator.

Example 2. SIMPL

One rather obscure language is especially pertinent in two ways. First, the Single Identity Micro-Programming Language (SIMPL) [9] is intended for designing microprograms. They, like hardware logic designs, must deal with the most basic gate-level operations and must provide for specification of timing and parallel processes. This language is intended to achieve machine independence by being a high level procedural (compiler) language.

Second, the SIMPL language is nearly unique with respect to control structures. The usual feature of procedural languages which provides that statements are executed substantially in the order written is avoided in SIMPL. A sequence of SIMPL statements might be:

A + B → C (1)

C * D → E (2)

A ↑ J → F (3)

In compiling this program segment for execution by perhaps a set of processors, statement (1) must be executed before statement (2), but statement (3) may be executed before or after either (1) or (2). This degree of freedom is detected in compilers for other languages (such as the IBM FORTRAN IV H compiler) by analysis of the variables appearing in the arithmetic expressions; the freedom is exploited in optimization of register assignments, for example. In SIMPL, however, the partial order among statements is explicitly indicated, due to adherence to the single

assignment rule of the language: no variable name may appear on the right-hand side (the "destination" of the assignment) more than once.

In a traditional compiler language, this would require allocation of a variable identifier, and perhaps a storage location, for each substitution statement's variable - a potentially great waste of storage. In the microprogramming application of SIMPL, however, the variable names are used to indicate states of a computation graph. Typically, a casregister is identified; say EXPA, the exponent portion of floating point register A: successive operations on this datum are denoted by assignments which produce variables whose names may be EXPA.1, EXPA.23, EXPA.15 in that relative order. Variable names are then typically composed of a data path name and a data path state number. The variable represents a state in the path of calculations on some given data. The sequence above might correspond to the following portion of a process graph:

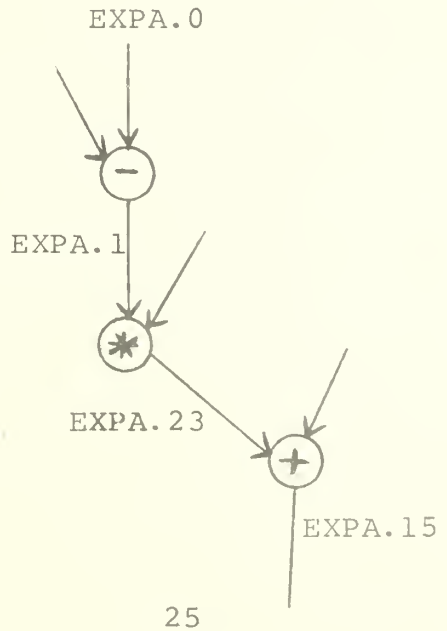


Figure 4.2
Control in SIMPL

The natural choice for a mapping which characterizes the progress through a computation process graph is the one which identifies the ordered state of each data path. Compiler language statements listed below in Table 4.1 such as goto are control statements in the sense that they specify relative order of respective sections of program.

Table 4.1 SIMPL language features

1. Data Operators

Arithmetic	Logical	Shift
+ addition	$\wedge$ AND	$\dashv$ left shift
- subtraction	V OR	$\vdash$ right shift
* multiplication	$\oplus$ exclusive OR	$\multimap$ left circular shift
/ division	$\neg$ NOT	$\circ\!\!-$ right circular shift
$\uparrow$ exponentiation		

Table 4.1 (cont.) SIMPL language features

2. Familiar control structures

Iteration

1. for a step b until c do s ;
2. while B exp do s;

Note: in the iterative body, s, of the iteration statements, exceptions to the single assignment rule were made. Only within the range of the iteration statement, such statements as the following are valid:

$$I + K \rightarrow I$$

Conditional

if B exp then s else s;

(The else s clause is optional)

Unconditional transfer

goto d;

goto a ($l_1, l_2, \ldots, l_n$); "computed GOTO"

I/O

read (a_1, r_1: a_2, r_2: ...);

write (a_1, r_1: a_2, r_2: ...);

3. Familiar interprogram control structures

procedure declaration

procedure NAME ($p_1, \ldots$);

procedure call

NAME (,,p_3, p_4,,$p_6 \ldots$);

NAME → REG1;

entry declaration

entry NAME;

exit NAME;

Example 3. AADC A and C

The AADC project has been responsible for bringing Navy interests to bear on many modern concepts in computer systems design. A particularly interesting facet of the AADC project, with respect to this control study, is the set of instructions for the Processing Element (PE) [3]. Although not an example of a compiler language, this hardware design "incorporates many characteristics found in the major Higher Order Languages" (sic) [3, p.1].

The PE is a general purpose serial processor containing a PMU (Program Management Unit), an AP (Arithmetic Processing Execution Unit), and a small TM (Task Memory). Instruction and operand fetching and program management instructions are executed by the PMU, leaving the AP free to compute concurrently, using operands and operators previously stockpiled in a "Q" by the PMU.

One of the important functions of a traditional compiler, the sequencing of operations in an expression, is in large part relayed to the AP. Non-commutative dyadic operators appear in both forms in the AP instruction set. That is, if x is a register and y is an operand, the AP can directly execute either x-y or y-x. In addition, "parenthetical control" can be used to temporarily defer execution of some operators until their precedents are complete.

The AP instruction set includes several forms each of (most of) the following operators:

dyadic arithmetic: add, subtract, multiply, divide, compare (numeric result), maximum, minimum

dyadic Boolean and logical: AND, OR, NOR, NAND, bit comparisons

monadic Boolean and logical: complement

transfer: transfer on one of a host of conditions, such as 'A $\geq$ 0' etc.

monadic: load, negate, absolute value, signum, floor, ceiling, square root, shift, store

array operations (1 and 2 dimensions): reduction, outer product, index, ravel, catenate, transpose, shape, take and drop, reversal, expand, compress, polynomial (and trigonometric functions)

The PMU instructions in the following classes may have more relevance to the control study. Indeed, some subsystems may contain PMUs without corresponding APs.

. Command Subsystem: send a word to an external destination (example commands which may be sent are to read or write a page or word)

. Initiate New Task: start up a task from a given large-storage address

. Scratchpad load and store

. Transfers (some are conditional on scratchpad contents).

- Execute
- Push and pop stacks of PMU accumulator, state registers and data
- Manipulate control bits of operands, internal timer, addressing mode,
- Halt, transfer to the executive
- Logical and arithmetic functions on words which are in or addressed by scratchpad locations

Example 4. FORTRAN

Table 4.2 diagrams the control structures corresponding to the statement types of IBM FORTRAN IV [7]. FORTRAN grew -- it was not designed -- as a language for FORmula TRANslation. It is interesting to note that, although most of the statement types correspond to the most elementary of control functions, the iteration loop is highly specialized. The peculiar features of this form of loop and the restrictive nature of the built-in data declarations make FORTRAN particularly cumbersome in some applications.

FORTRAN Item	Related Control Function
Declarations DIMENSION, COMMON, BLOCK DATA, EQUIVALENCE, DATA, INTEGER, REAL, LOGICAL, COMPLEX, FORMAT, constants and literals NAMELIST	none (data definitions)
FUNCTION, ENTRY, RETURN EXTERNAL, END	program linkage
Assignment variable = expression	evokes a (possibly complicated) data operation as specified by "expression:" this may involve calling other routines ("function subprograms")
CONTINUE	null data operation; can be labelled as a step in the process
CALL	evoke a separately compiled routine (a "subroutine subprogram")
PAUSE	suspend operation until operator intervenes
STOP	terminate execution

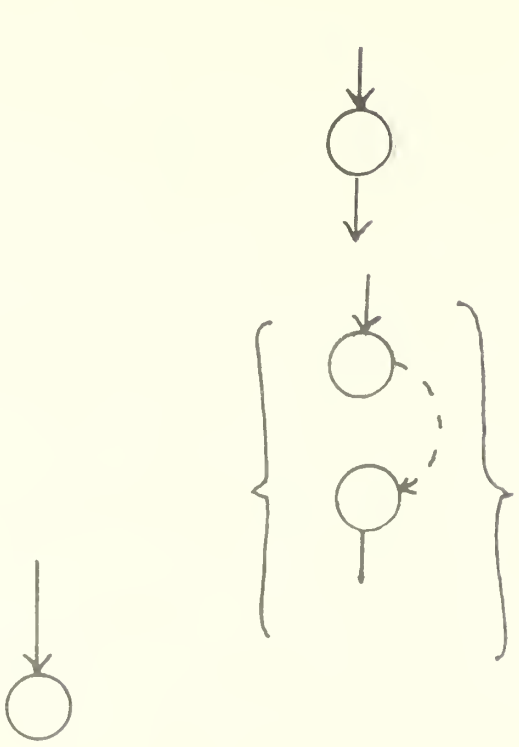
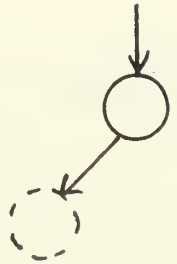


Table 4.2 CONTROL FEATURES IN IBM FORTRAN IV

Unconditional GO TO
GO TO x



program transfer: alter the normal sequential flow

multiple branches:
Computed GO TO
GO TO $(x_1,x_2,\ldots,x_n)$, i

i indexes the label from the list of possible next nodes

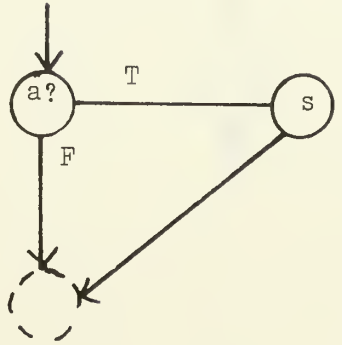
Assigned GO TO
ASSIGN i TO m
GO TO m,$(x_1,\ldots,x_n)$

m is ASSIGN'd to identify which node is next

Arithmetic IF
IF (a) x_1,x_2,x_3

next node is x_1, x_2 or x_3 according to whether expression a is > 0, = 0 or < 0

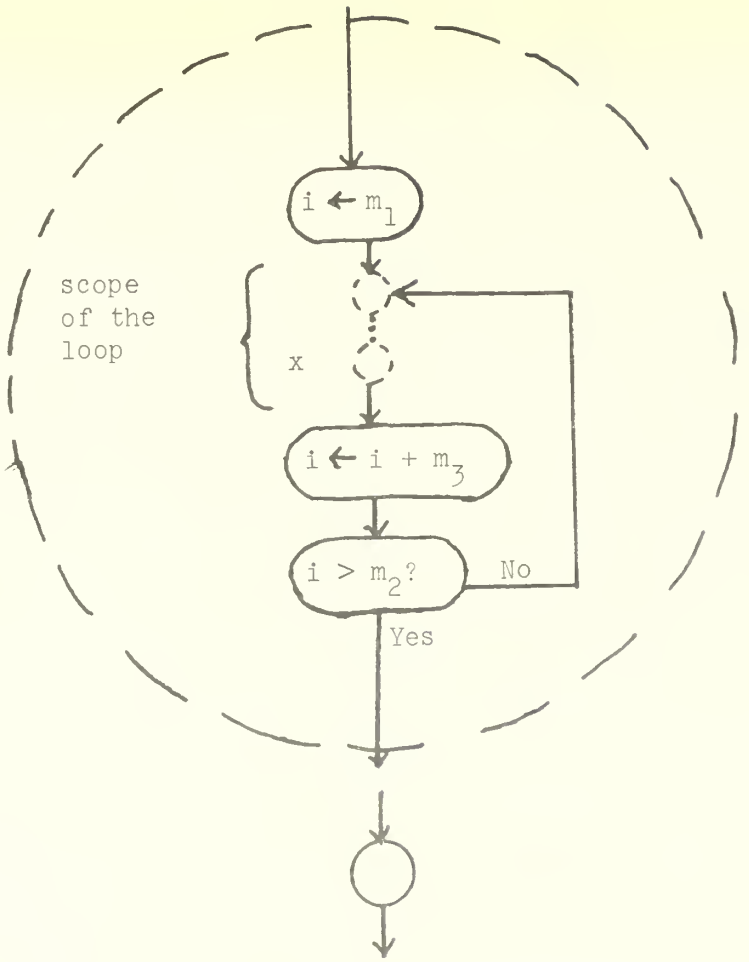
Logical IF
IF(a) s



The single statement s is executed only if logical expression a is "TRUE". Control then sequences to the next statement in the normal fashion.

DO statement
DO x i = m_1, m_2, m_3

"DO loop"



an iteration loop with several peculiar features, among them:

- m_i are unsigned integer variables or constants only
- the test is performed <u>after</u> the scope is executed
- transfers <u>out</u> of a loop are permitted, but not <u>into</u> a loop (in general)
- the indexing parameters (i, m_1, m_2, m_3) are not changeable within the loop, nor is i defined upon completion of a loop

I/O statements
READ, WRITE, PUNCH,
PRINT, ENDFILE, REWIND,
BACKSPACE
DEFINE FILE, FIND

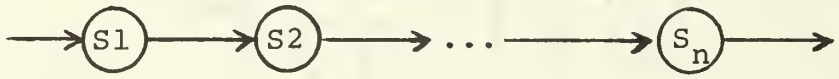
evoke input - output routines

Example 5. ALGOL

Based on the design collaboration of an international group of experts, ALGOL enjoys the advantages of having a formally defined syntax and being based on the contructs desired for numerical mathematics rather than on the architecture of a particular type of computing machine. Because of this, ALGOL tends to better illustrate the essential flavor of compiler language programming, and is a better vehicle for teaching these ideas than a language such as FORTRAN. The principle control structures of ALGOL which are illustrated below are abstracted from a description of PASCAL, an extension of ALGOL 60 [11].

5.1 Compound statements

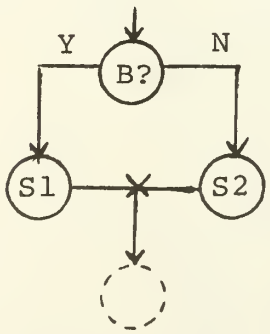
begin S1; S2; ...; S_n end



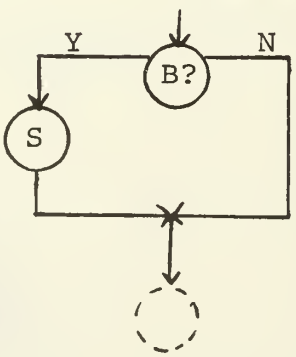
A compound statement can be used in place of any of the appearances of S_j below:

5.2 Conditional statements

if B then S1 else S2

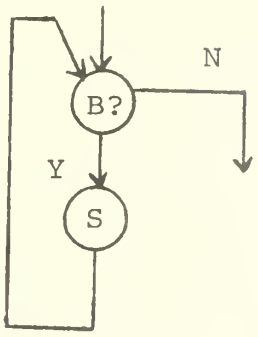


if B then S

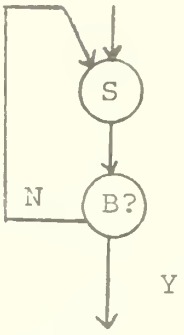


5.3 Iterative (repetitive) statements

while B do S

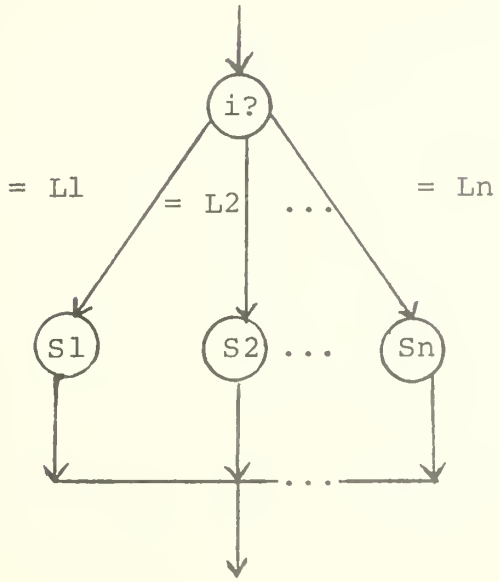


repeat S until B



5.4 Selective (case) statement

case i of L1: S1; L2: S2; ...; Ln: Sn end



Example 6. PL/I

The PL/I language was created by IBM users frustrated with the ad hocracy of FORTRAN. Although the compilers and the object programs for implementations on various machines may be less efficient than desirable, there are a number of advantages. Many of the differences lie in the data definition facilities: PL/I allows structured data definitions, dynamic storage allocation, some data types not available in FORTRAN, and a richer set of I/O operations.

The control features, as compared to FORTRAN, have differences in two respects. First, there are mechanisms for creating and communicating with asynchronous procedures (independent, parallel tasks). Second, program flow control structures similar to those in FORTRAN are generalized, allowing more flexibility. The list of PL/I control structures below displays most of the differences in basic form without showing all of the details [6]. Figure 4.3 diagrams the structures in terms of control state sequencing.

1. process control

   1.a. task creation

   CALL task (pars) EVENT (eventname) invokes task "task" which has been declared as an asynchronous process and establishes "eventname" as a variable which can be checked to determine whether or not "task" is finished.

1.b. task deletion

Tasks destroy themselves by completing (RETURN or END)

1.c. task synchronization

One task checks on the progress of another by waiting for it to finish:

WAIT (eventname)

or by testing the logical (true/false) value of the condition:

IF COMPLETION (eventname) THEN ....

2. FORTRAN - like control structures

The extra generality in the following statements arises partly from the fact that "group" may be a single statement or a block of statements. Below, "label" is a (possibly subscripted) execution - time variable and "expr" is an expression which is a generalized logical expression.

2.a.

GOTO label

Similar to FORTRAN, except the labels are variables and may have computed values.

2.b.

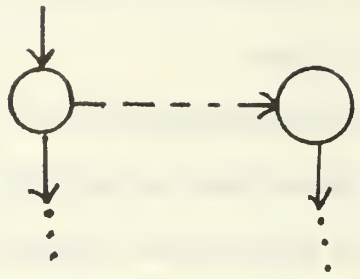
IF expr THEN group ELSE group;

arbitrarily complex paths in either branch.

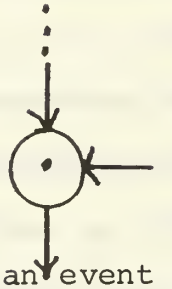
2.c.

label DO index = initial TO final group END label;

The closest correspondent to the FORTRAN DO loop is again more flexible.



1a (new task)

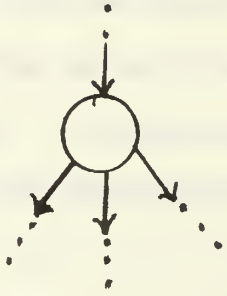


1c WAIT

does not proceed until event has occurred

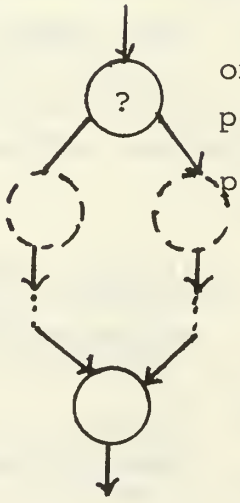
event

and COMPLETION : allows an event name to be tested as data



2a GO TO

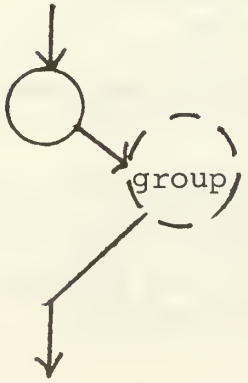
number of possible branches depends on declarations and computations



2b IF ... THEN ... ELSE ....

one or the other, possibly complex path

2c DO



defines a block and controls iteration

Figure 4.3 Control Structures in Pl/I

5. Applications

Several applications for the view of control adopted for this report present themselves. Each application area is a whole field of study in itself; only the fact of relevance is noted below. In most cases, the most immediate effect of applying the present view of control structures would be to unify work which has already been done in the diverse application areas. Further developments can be expected to improve the efficiency of procedures and designs within the areas.

5.1 Building controls for digital systems

Heretofore, the design of digital systems has generally been explained entirely in terms of building the data operators. It appears that we are ready to move into a new level of design, where data and control operators share in importance.

5.2 Compiler optimization

To some degree, the construction of a program can be automated. This is particularly true in the case of something like the FORTRAN compiler, which translates from one specific language (FORTRAN) to another (machine language). In particular, the IBM level FORTRAN IV G compiler attempts to increase execution speed by changing the order in which the machine steps implied by the FORTRAN program are executed.*

*Note that some execution errors may be caused if the original programmer does not take steps to avoid certain "simplifications." Moral: a smart compiler may outsmart the user in a language which is sufficiently awkward.

Many of the attempts at methods to optimize code can be re-viewed as attempts to optimize the pattern of control structures. On the other hand, the optimization problem can be broadened for the case of hardware design to include choosing the best control structures for a program or class of programs.

5.3 Automatic microprogramming

This topic includes variously the choice of a convenient microprogramming structure (a set of control structures for the microprogrammed interpreter) or the construction of the microprogram for a specific computation. Elements of the topics in 5.1 and 5.2 enter here. Note especially Example 2 in Section 4.

5.4 Operating systems

It requires mostly a change in the level of description to make much of the work in operating systems relevant to the current discussion of control structures. Given a protected multiprocessing system, probably with virtual memory, the creation and synchronizing of processes and the management of resources can be viewed in terms of control structures. The key here is the selection of a small enough amount of information to describe the control state. The usual practice seems to define the state of a process very much according to the hardware implementations which are critically important to the operating system, and to include an immense amount of detail, in order to provide completeness. For example, it has been maintained that the

complete virtual memory map of a process is necessary for its description [10]. It may be more appropriate to find a level of abstraction where such information may be regarded as data to be manipulated by data operators, and where there is a more convenient set of state variables. The design, measurement and evaluation of operating systems might then be enhanced by a concentration upon the relevant control structures.

REFERENCES

[1] G. G. Bell, A Newell, Computer Structures: Readings and Examples, McGraw-Hill, 1971, p. 134.

[2] A. W. Burks, H. H. Goldstine, J. von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," Quoted from reference [1], p. 111.

[3] A. Deerfield, et al, "Final report for AADC arithmetic and control logic design study, Part III, Preliminary programmers reference manual," BR-7162-3, contract N62269-72-C-0023, Naval Air Development Center, Warminster, Pa., December, 1972.

[4] W. J. Dejka, J. W. Wasilewski, D. Harrison, "Implementation of mathematical functions," prepared for the NACON '73 meeting in Dayton, Ohio, May, 1973.

[5] J. J. Horning, B. Randell, "Process structuring," Computing Surveys, 5, 1, March, 1973, pp. 5-30.

[6] IBM, "IBM System/360 Operating System PL/I Language Specifications," Form C28-6571, 1965.

[7] IBM Systems Reference Library Form C28-6515, "IBM System/360 FORTRAN IV Language"

[8] V. M. Powers, "Functional program modules (FPMs) and digital systems design," US Naval Postgraduate School report NPS-52PW72071A, 20 July 1972.

[9] M. Tsuchiya, C. V. Ramamoorthy, "Design of a multilevel microprogrammable computer and a highlevel microprogramming language," Tech Report 135, Information System Research Laboratory, Electronics Research Center, The University of Texas at Austin, August 15, 1972.

[10] R. W. Watson, Timesharing System Design Concepts, McGraw-Hill, 1970.

[11] Niklaus Wirth, Systematic Programming: an Introduction, Prentice Hall, Inc., 1973.

[12] "Small computer handbook," Digital Equipment Corporation, 1968.

DISTRIBUTION LIST

Defense Documentation Center Cameron Station Alexandria, Virginia	12
Library Naval Postgraduate School Monterey, California 93940	2
Naval Electronics Laboratory Center Research Library, Code 6700 San Diego, California 92152	2
NELC Mr. W. J. Dejka, Code 4300 San Diego, California 92152	8
NELC Mr. S. Snyder, Code 4000 San Diego, California 92152	1
NELC Mr. H. T. Mortimer, Code 0220 San Diego, California 92152	1
NELC Mr. M. Lamendola, Code 5200 San Diego, California 92152	1
NELC Mr. W. Loper, Code 5200 San Diego, California 92152	1
NELC Dr. R. Kolb, Code 3300 San Diego, California 92152	1
NELC Mr. E. E. McCown, Code 3100 San Diego, California 92152	1
NELC Mr. H. F. Wong, Code 3200 San Diego, California 92152	1
NELC Dr. P. C. Fletcher, Code 2000 San Diego, California 92152	1

Dean of Research
Naval Postgraduate School
Monterey, California 93940 2

Professor Sydney R. Parker
Chairman, Dept. of Electrical Engineering
Naval Postgraduate School
Monterey, California 93940 2

Professor G. L. Barksdale, Jr.
Chairman, Computer Science Group
Naval Postgraduate School
Monterey, California 93940 2

Professor U. R. Kodres
Department of Mathematics
Naval Postgraduate School
Monterey, California 93940 1

Professor G. A. Myers
Department of Electrical Engineering
Naval Postgraduate School
Monterey, California 93940 1

Professor G. H. Syms
Department of Mathematics
Naval Postgraduate School
Monterey, California 93940 1

Professor V. M. Powers
Department of Electrical Engineering
Naval Postgraduate School
Monterey, California 93940 8

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS-52PW73081A	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) CONTROL STRUCTURES IN DIGITAL PROCESSES		5. TYPE OF REPORT & PERIOD COVERED Technical Report 1973-74 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) V. Michael Powers		8. CONTRACT OR GRANT NUMBER(s) Work Request WR-2-9104
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE 29 August 1973 13. NUMBER OF PAGES 48
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

digital control
control structures
control states
digital process control

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The control space of a digital process can be viewed as a projection of the state space of the processor. This state space may be an interpretation of some underlying (perhaps physical) processor's state space. A control operator is a projection of a process step: the portion which specifies the "next control state". A set of elementary control structures is defined and used as a common basis for comparing the control structures in a microcomputer and

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601
(Page 1)

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. several programming languages. The relationship of this view of control to several areas of computer science research is noted.